Binding :- (Linking)

A binding is an association of an attribute of a program entity with a value.

→ Each program entity has a set of attributes associated with it.
e.g. A variable has attribute like type, scope, dimentionality, memory address, etc.

→ Binding time is a time at which binding of attributes with a value is performed.

Types of binding :-

int a[10];

1. **Static binding :-**
A static binding is a binding performed before the execution of a program begins.

Ex:- Consider memory binding which is an association between the memory address attributes of a data item & the address of a memory area.

In static memory allocation, memory is allocated to a variable before program execution begins. Typically performed during compilation.

No memory allocation & deallocation is performed during execution

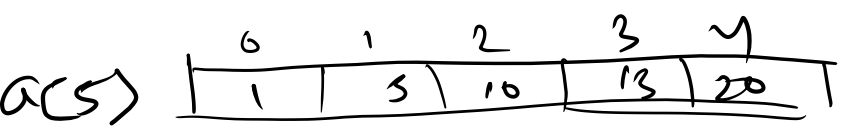
2. Dynamic Linking/Binding:

A dynamic binding is a binding performed after execution of a program has began.

→ In dynamic memory allocation, memory binding are established & destroyed during execution of a program.

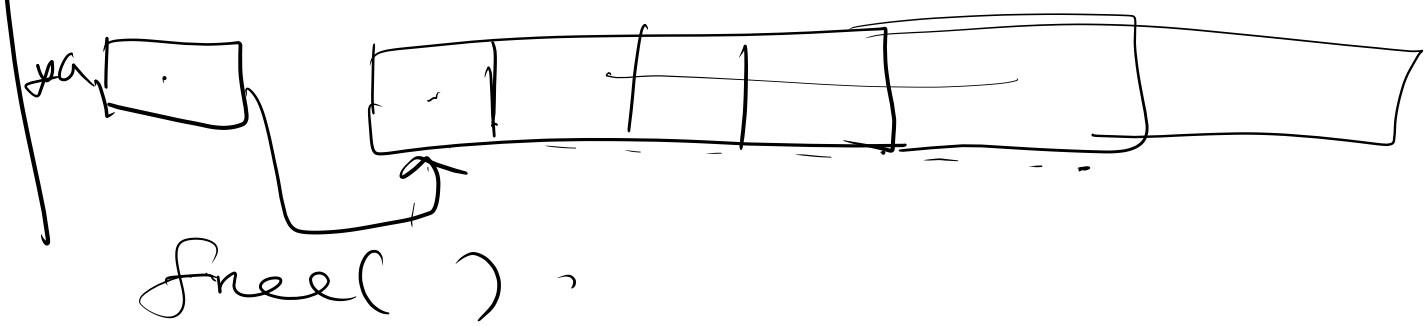
Static binding

```
int a[5];
a[5] = {1, 5, 10, 15, 20};
```

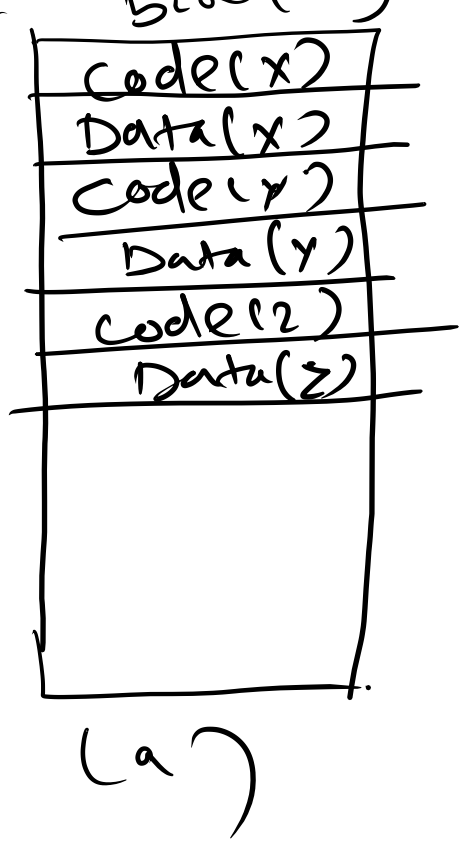


Dynamic binding

```
int *a;
a = (*int) malloc (5 x sizeof(int));
```

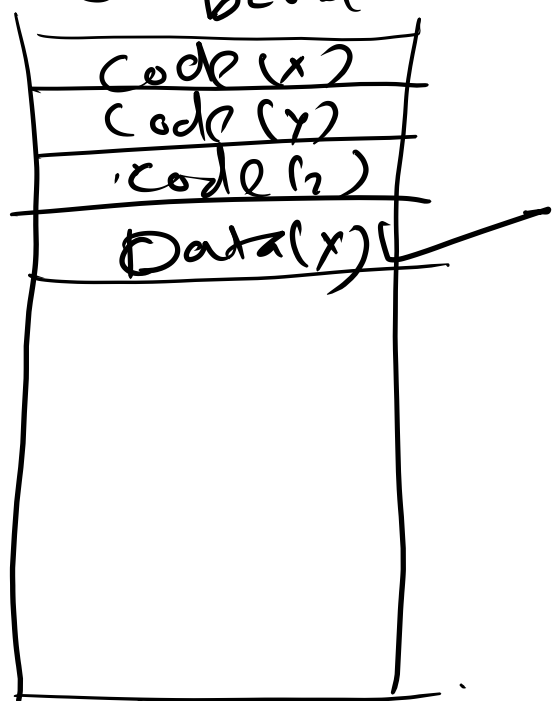


Static binding



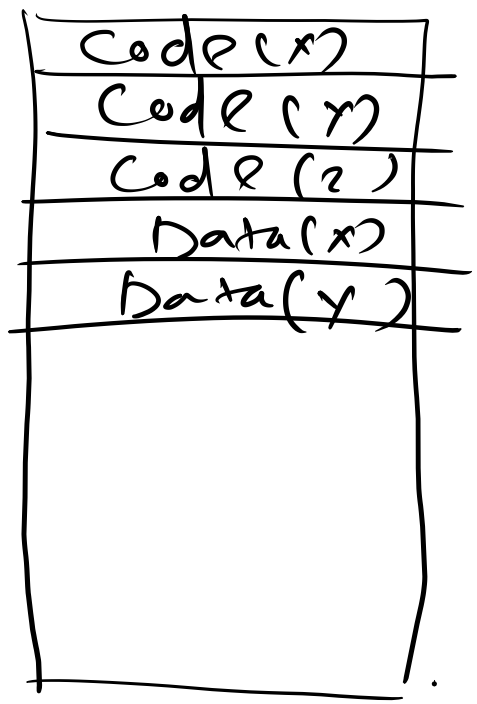
(a)

Dynamic binding



(b)

when x call & y



(c)

Let x, y & z are 3 program units

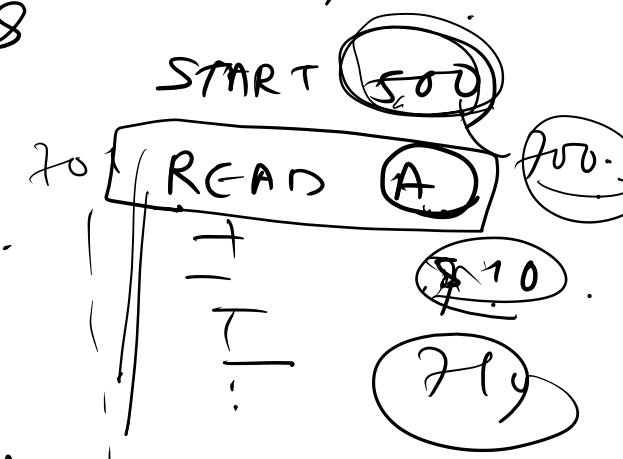
Translated, Linked and Load Time Address :-

→ The origin of a program may have to be changed by linker or loader for some reasons like if same set of translated addresses may have been used in different object modules or the operating system might require that the program should execute from some other location.

→ The change of origin leads to change in execution start address and in the address assigned to symbols.

Generally there are 3 address terminologies are they.

1. Translated time address :- address assigned by translator.
2. Linked address :- assigned by linker.
3. Load time address :- assigned by loader.

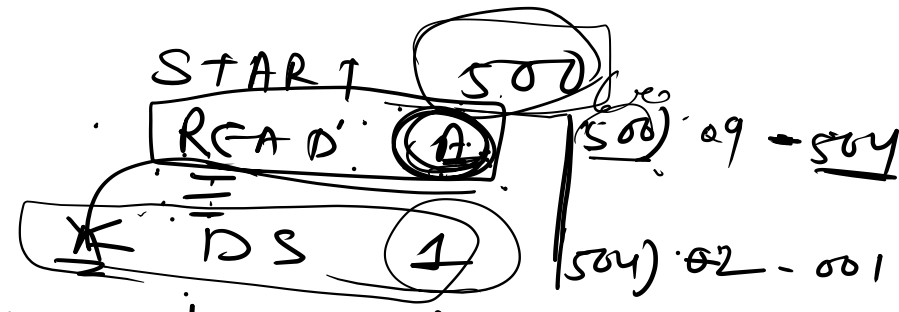


Translated origin — refer to the address of the origin assumed by the translator.
(In assembly language, START or ORIGIN is used as origin specifying statement)

Linked Origin - refer to the address of origin assigned by linker while producing binary program

Load Origin - refer to the address of origin assigned by loader while loading the binary program for execution.

Relocation & Linking Concept :-



Program relocation :-

→ If instructions in a program refer to absolute address i.e. the program assumes it's instructions and data to occupy memory with specific addresses, such programs are called address sensitive program. — address sensitive instn
— Constant

→ An address sensitive program P can execute correctly only if the start address of memory area allocated to it is same as it's translated origin.

→ Program relocation is a process of modifying the addresses used in the address sensitive instructions of a program such that program can execute correctly from designated area of memory.

If linked origin ≠ translated origin, relocation must be performed by linker.

If load origin ≠ linked origin, relocation must be performed by loader

Note :- Generally, the task of relocation is done by linker, not by loader.

So, we assume that load origin = link origin. Such loader are called absolute loader.

Performing Relocation :-

In order to perform relocation of address sensitive instructions, we must find out relocation factor which is the difference between linked origin and translated origin.

> Relocation factor = linked origin − translated origin

→ can be +ve, −ve or zero.

Consider a statement that uses a symbol 'S' as an operand & translator generates it's address as

> translated address = translated origin + displacement (offset of S)

> Linked address = Translated address + relocation factor

```
                                        M/C Code.
START   500
ENTRY   TOTAL
EXTRN   MAX, ALPHA
READ    A                               500) 09 0 540
                                        501)
LOOP
        MOVER   AREG, ALPHA             518) 04 6 550
        BC      ANY, MAX                519) 06 6 000
        BC      LT, LOOP                538) 06 1 501
        STOP                            539) 00 0 000
A       DS      1                       540) 01 - 001
TOTAL   DS      1                       541) 01 - 001
        END                             542)
```

Fig: Program P1

Size = 42 words

Ex:- Consider Program P

In program P, there is an instruction READ A. This is an address sensitive instn because it uses address 540 (address of symbol A)

As per program P, translated origin = 500.

If linked origin = 700, then A would have linked address calculated as follows;

Relocation factor = linked origin − translated origin
= 700 − 500
= 200

Translated address of A = translated origin + offset
= 500 + 40
= 540

Linked address of A = Translated address of A + relocation factor
= 540 + 200
= 740.

Public and External References :-

→ A program unit P, interacts with another program unit Q by using address of Q's instⁿ and data in it's own instⁿ. for this P & Q must contain Public definition and external references as follows:

i) Public definition (PD) : A symbol defined in program unit may be referenced in other program unit.

ii) External References : A reference to symbol which isn't declared in the program containing references.

→ The ENTRY statement lists the public definition of a program unit.

→ The EXTRN statement lists the symbols to which external references are made in program unit.

Note :-

Assembler doesn't know the address of an external symbol, hence it put

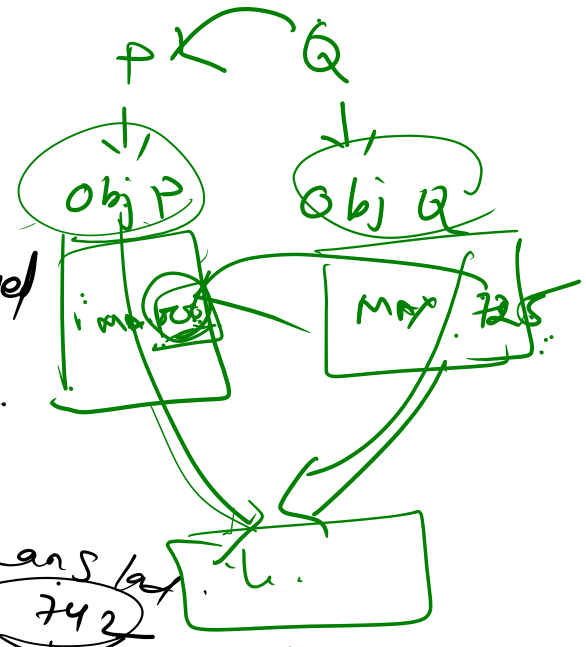
0 in the address field of corresponding instⁿ.

If EXTRN doesn't exist, the assembler would have tagged references to MAX & ALPHA as error.

Linking :-

Linking is the process of binding an external reference to the correct link time.

→ An external reference is said to be unresolved until linking is performed for it. It is said to be resolved when linking is done.



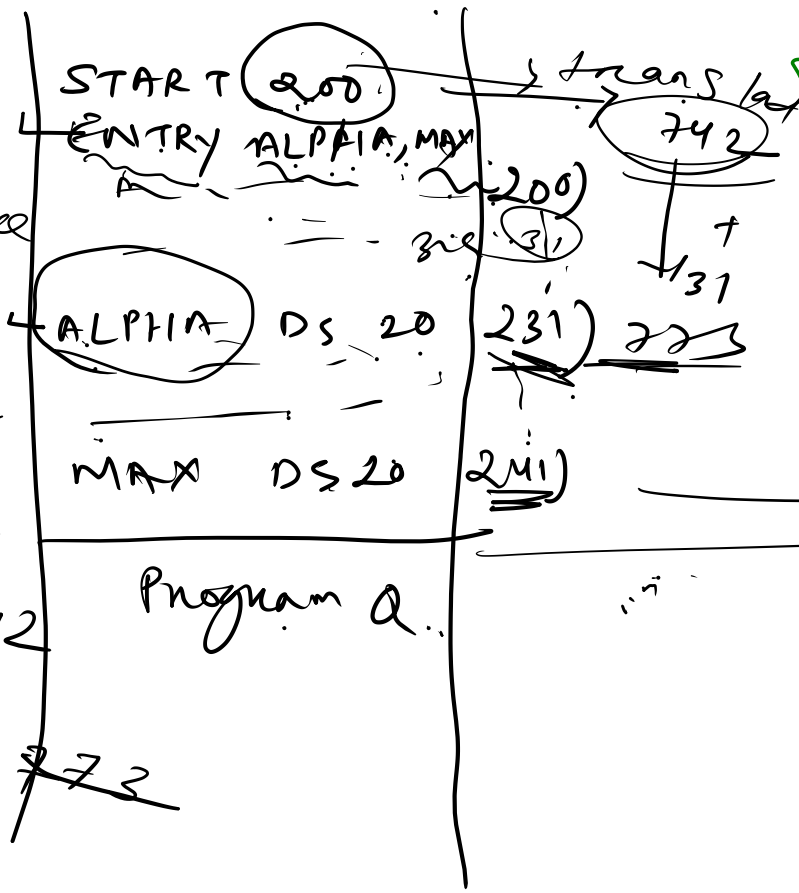
Let the program unit P_1 be linked with the above program P_2

→ P_1 contains an external reference to symbol ALPHA which is a public def in P_2. with translation time address is 231.

→ If link origin of P_1 is 700 and size of the P_1 is 42 words then the link origin of P_2 is 742

Link time address of ALPHA

$= 742 + [(231 - 200)] - 742 + 31 = 773$



Binary Program :-

→ A binary program is a m/c language program comprising a set of program units, SP such that $\forall P_i \in SP$

→ P_i has been relocated to the memory area starting at it's link origin and linking has been performed for each external reference in P_i

→ To form a binary program from a set of object module, the linker has to be invoked using the command

Linker <link origin>, <object modules>, [<execution start address>]

Here <link origin> specifies memory address to be given to the 1st word of the binary program

<execution start address> is a pair

< program unit name, offset of ep program unit >

Object Module :-

→ The object module of a program contains all informations necessary to relocate and link the program with other programs.

→ Object modules having 4 components :-

1. Header :- The header contains translated origin, size and execution start address of the program.
2. Program :- M/c language program corresponds to the original program.
3. Relocation Table (RELOCTAB) :- This table describes the set of instructions requiring relocation in program & each entry has a single field.
4. Linking Table (LINKTAB) :- This table contains information about public definition and external references of the program.

LINKTAB has 3 fields :-

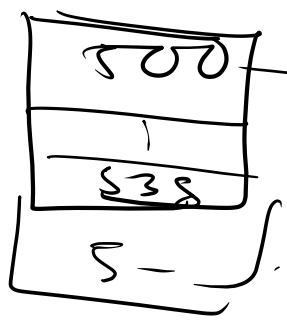
a) Symbol : Symbolic Name
b) Type : PD/EXT indicates Public Definition / External Reference.
c) Translated address :- for public def^n, this is the address of 1st memory word allocated the symbol. For external reference, this is the address of memory word which is required to contain the address of the symbol.

Ex: Let P_1.

Header) Translated origin = 500
Size = 42
Exec. start add :- 500.

MIL

RelocTAB :-



Linked TAB

LINKTAB of P_1

Symbol	Type	Translated address
ALPHA	EXT	518
MAX	EXT	519
TOTAL	PD	541

Relocatability of Program:-

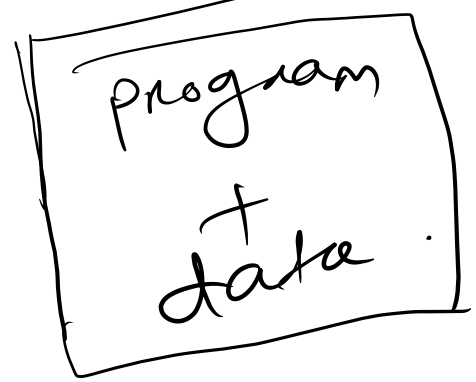
1. Non-relocatable Program:-

→ It is one which can't be made to execute in any area of memory other than the area starting on it's translated origin.

→ A non-relocatable program is a term defined as a result of address sensitivity of present code which is lacking information of:

a) which part of the program are address sensitive
b) in which manner

e.g. .com file of MS-DOS.

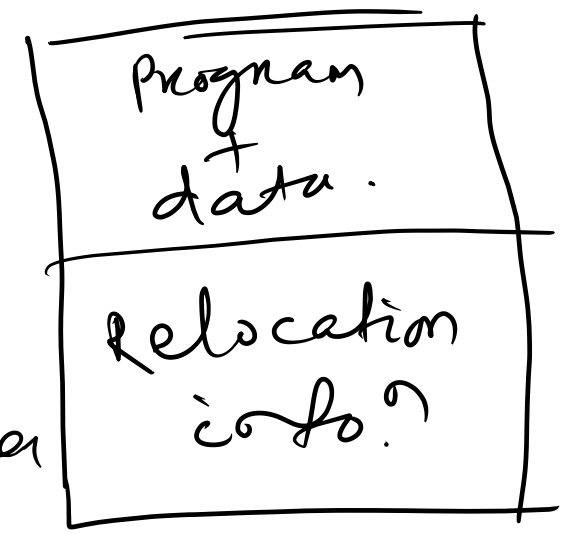


2. Relocatable Program :-

→ A relocatable program is a unit which has load address & information used for relocation purpose.

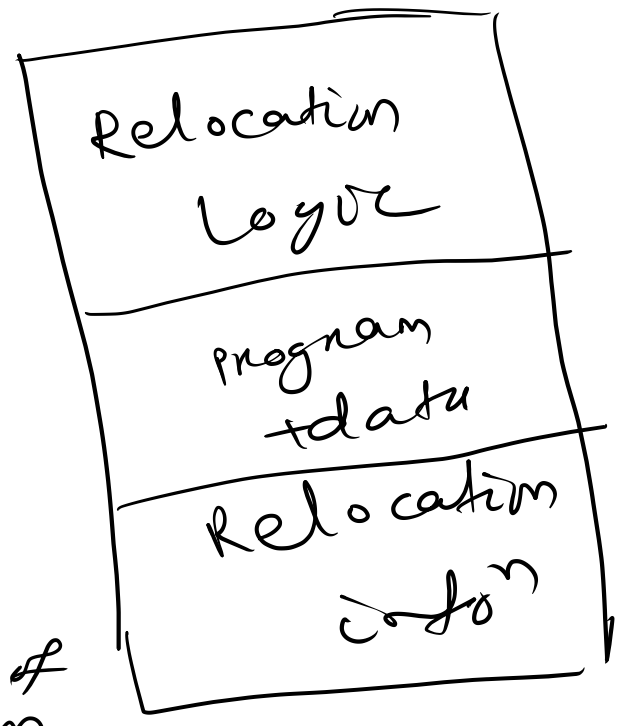
→ These programs contain info.ⁿ about address sensitive instruction.

→ Using relocatability information, program can be loaded at different location other than translation time address.



e.g. .exe file on DOS.

3. Self-relocatable program :-



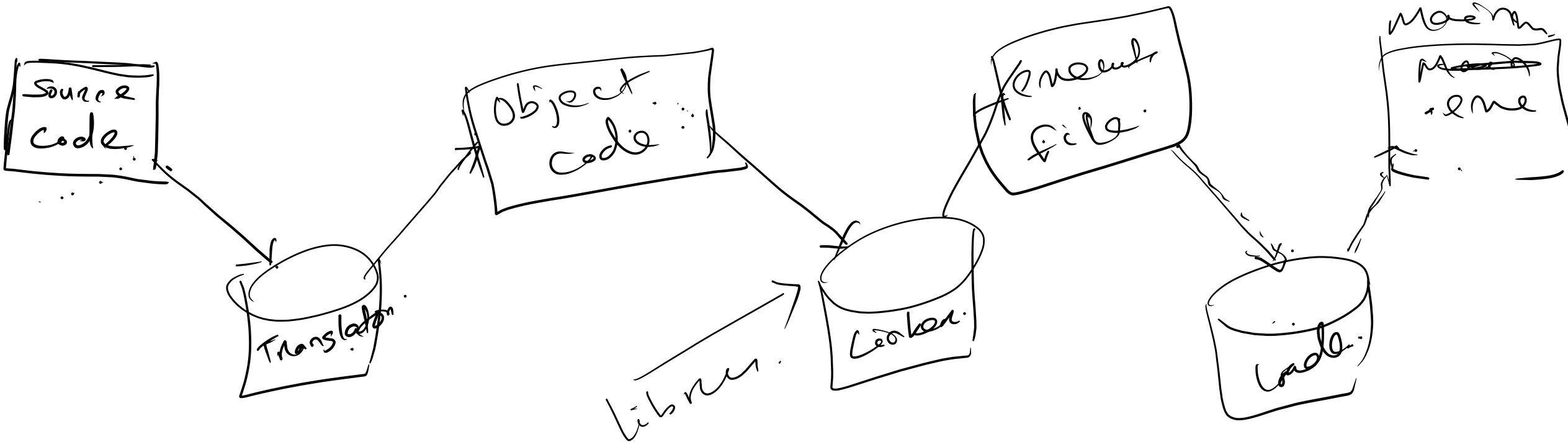
→ A program which performs relocation of it's own address sensitive instruction is called as self-relocating program.

→ A task of relocating itself is on the relocation logic. This code also has a table of information related to address sensitive instruction called as relocation information.

→ A start of relocating logic address is always specified as execution start address of any program. By making use of relocation logic and all information related to address sensitive instruction, it performs relocation by itself.

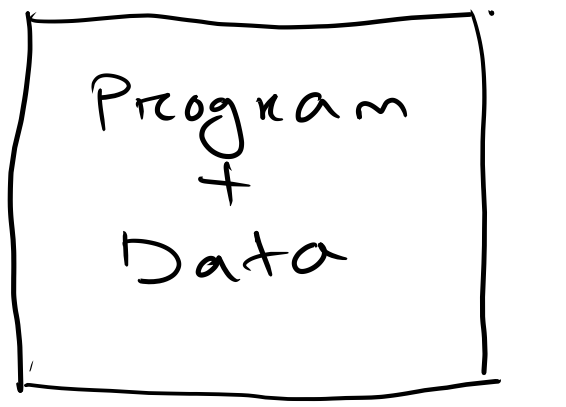
→ After relocation of program, execution control is transferred.

→ Self-relocating programs can be executed at in any area of memory.



⟹ : Data flow

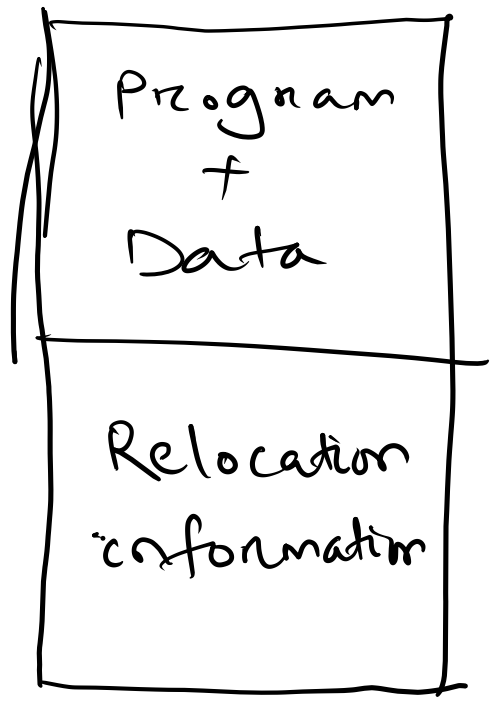
⇢ : Control flow.



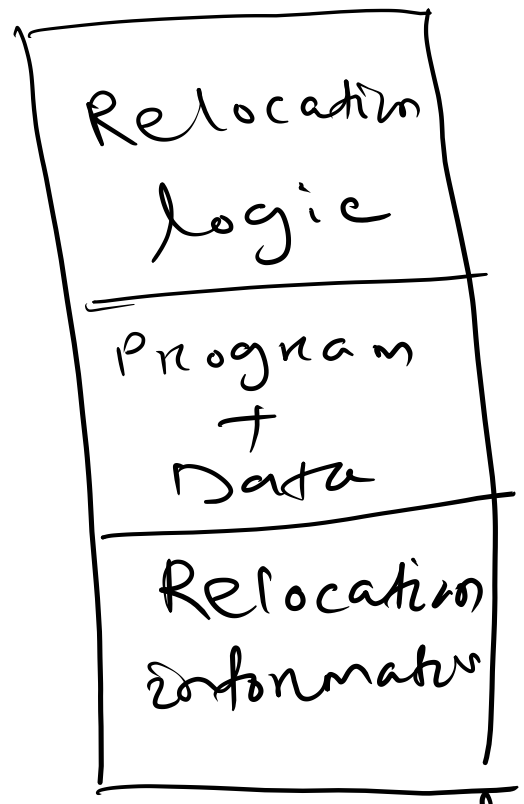
[Non-relocatable]

- com of DOS
- some of MSDOS

Single user OS



[Relocatable program]



[Self-relocating program]

compiler: TC, FORTRAN, COBOL is multi-user system contains code for self-relocation.